Addendum to C=Hacking #20

```
I couldn't resist, and tried something out (see attachment). It works!!! :-)

In fact, when I wrote the last letter I didn't know that I found something
useable, just had some ideas - I felt that I'm at the right place. When I
read
C=H 20 this morning and read your comment about the Test bit (from the PRG),
I
knew that it must work. All I had to do is then to put this idea into code.

The whole idea is about starting the pulse by software, and then having the
SID turn it back to 0 after a time.

Is it possible? ...The keys are the Test bit (the SID wave counter can be
reseted anytime), the pulse width register, the wave counter and the SIDs
way
of generating pulse wave. (Ie. the pulse wave is high, as long as the wave
counter is less than the value in the pulse width register).


Check this algorithm:

- Init: volume at max, voice 1 sustain level max, start attack. Freq is
selected well (=$4000), so the wave counter is incremented by 4 every
processor clock cycles.

Loop:
- load next sample value, and put it to the pulse width low register ($d402;
ensure that $d403 is 0).
- Set test bit, and clear test bit (counter reset).
- Increase sample pointer, some delay, then loop. The delay must be 64 clock
cycles + the time while the Test bit is kept set (4 cycles if using STA
$d404
: STX $d404 immediately with pre-loaded values).

What will happen? The 8-bit sample value is put directly to the pulse width
register (MSBs of the pulse width register are cleared!...). The wave
counter
is started (release test bit), and it increases 4 by every CPU cycles (=
counts 256 in 64 cycles). After some time, the counter will reach the value
in
the pulse width register. This happens in exactly after (8-bit sample value
/
4) cycles, because of the above. In this cycle (or the next?...) the SID
turns
its pulse output to 0. Voilá!

One must just make sure that the loop length in cycles matches the above
```

```
conditions, and then it runs like hell... Since it does exactly the same on
the SID as the other (bit-banging) way, it just does it with some hardware
help, there's also no problem with the 4khz maximum barrier (since the
oscillator is reset every loop).

With little enhancement, it's possible to write an about 7.5 bits player for
a
stock C64 by this method. This is what you find in the attachment... The
idea
is using all the 3 channels simultaneously. A slightly increased sample
value
is written to the three pulse width registers, so the oscillators will
finish
the duty cycle one processor cycle later, when there's a carry between
bits(0,1) to the MSBs.

The replay freq is the CPU clk / 68 (~15khz). 64 cycles (variable duty
cycle)
+ 4 cycles (constant duty cycle because of the reset time - no problems with
that, it doesn't change (just gives a small constant DC...)).

By similar methods, it should be possible to write a sample player with
higher
PWM freq (with less resolution of course, but eliminating this still audible
whistling).

(I tried using the filter to reduce it, but it sounded so bad that I left it
out. It clicked like hell. The FETs got saturated.)

See the attachment, and the binary. I think the sample sounds pretty good :-
).
(The cut is from 'Greece 2000' by Three drives on a vinyl).

(Another idea that popped up in my mind: since the TED sound generator can
also be reset, I could probably translate this idea to the Plus/4 :-O ).

Best regards,

Levente
```